Js特殊的地方：

可以动态的添加属性，解释性语言的特点。

Js包含的几个部分：

核心：ECMAScript js语法标准

文档对象模型：DOM

浏览器对象模型：BOM

Window和document区别：

Function中this的指向问题：

1. 预编译this🡪window
2. 谁调用this就指向谁
3. Call apply 可以改变this的指向
4. 全局this 🡪window

访问对象属性：obj.property 🡪 obj[“property”]

obj.hasOwnproperty(“property”)和property in obj判断property是否是obj对象的属性

但是两者有区别，property in obj判定是否是obj的属性或其原型链上是否有这个属性

A instance of B判断A的原型链上是否有B的原型

判断一个对象是 {} 还是 []

方法1： {}.constructor->object [].constructor->Array

方法2： [] instance of Array->true {} instance of Object ->false

方法3： Object.prototype.toString.call([]) -> "[object Array]"

Object.prototype.toString.call({}) -> "[object Object]"

**继承**

继承的几种方式：

1. 原型链继承： Son.prototype = new Father()，Father的属性被Son.prototype继承，引用类型属性被Son对象实例共享
2. 借用构造函数继承（经典继承）：在子类中调用父类构造函数，该方法避免了引用值共享，但是每次创建实例都要调用一下父类构造函数
3. Object.create(obj)函数返回一个对象，并将该对象的构造函数的原型赋值为obj，所以会有引用值共享的缺点
4. 圣杯模式

继承的模式（现在经常使用的一个模式：圣杯模式）：

function inherit(Target, Origin) {

function F() {} //创造构造函数，当作中间量

F.prototype = Origin.prototype; //共享原型

Target.prototype = new F();

Target.prototype.constructor = Target; //把构造函数还原

Target.prototype.uber = Origin.prototype; //把Target最终继承自的构造构造函数v保存起来，方便访问

}

**圣杯模式相对于原型链继承的优势： 原型链继承模式Son.prototype = new Father()，son不仅继承了父类原型上的方法，而且继承了父类的属性，这样不方便子类属性的初始化，所以通过圣杯模式中中间函数与父类共享原型链来隔绝私有属性的继承，同时不干扰原型上方法的继承**

深度克隆：(被克隆的属性改变不会影响克隆的对象)

function deepClone (target, option) {

var copy, src;

if(option != null) {

for(var prop in option) {  
 src = target[prop];

copy = option[prop];

if(copy && typeof copy == “object”){

if(Object.prototype.toString.call(copy) == “[object Array]”){

src = src ? src : [];

}else{

src = src ? src : {};

}

target[prop] = deepClone(src, copy);

}else{

target[prop] = copy;

}

}

}

return target;

}

**ECMAScript---------核心**

函数定义方式：

1. var func = {}
2. var func = new 类型()
3. var func = 类型.create(原型)//此处的原型可以是构造函数或者null,所以不是所有函数都有原型

！！！函数中的属性分为私有属性(var name)和共有属性(this.name),私有属性无法通过函数的实例对象来访问，而共有属性可以被访问。

1．数据类型：基本类型和引用类型

基本类型：

Number：数字型，值为数字

Boolean：bool型，值为true 或者 false

Null：表示空，只有一个值。null

String：字符型，值为字符

Undefined：表示没有赋值

引用类型：（object）

理解为类，他的值为对象。

数据类型转换：

隐式转换：加减乘除符号，可将字符型转换为number型，但是undefined只能转化为NaN（不是任何值，但也是number型）

加号比较特殊，任何类型与字符相加都会变成字符型

显式转换：parseIInt(),parseFloat()

2.对象：

对象的属性：

**Prototype**:对象的原型，用来实现继承功能的关键对象，prototype必须赋值为实例对象，不能是函数对象(\_\_proto\_\_也是，因为\_\_proto\_\_指向的是函数的prototype，就是说他们指向一个对象)

Class:描述对象类型的字符串值，使用typeof操作符返回的结果

Extensible:设置是否允许对象扩展属性

Get:返回属性值，返回属性名的值

Put:设置指定的属性名的值  
Canput:返回指定属性是否可以通过put操作设置值  
Hasproperty:返回对象是否有指定的值

Delete:从对象中删除指定的属性值

DefaultValue:返回对象的默认值

Construct:可以通过new操作符创建一个对象。实现了这个内部属性的对象被称为构造函数

Call:执行关联在对象上的代码，通过函数表达式调用。

HasInstance:返回给定的值是否扩展了当前对象的属性和行为

Constructor:构造器，返回对象的构造函数

属性的特性：

Value:属性的值

Writable:是否为只读

Enumerable:是否能用for…in…循环来枚举属性

Configurable:是否可以用delete删除该属性

Get:设置一个getter访问器

Set:设置一个setter访问器 //不能访问属性自身会造成死循环

内置对象：

Global：全局对象

1：无constract属性，所以无法通过new操作来当作类来调用

2：没有call属性，所以他也无法像函数那样被调用

3：在浏览器中global对象就是window对象。当程序开始运作时，global对象已经被系统创建，并且他的属性也都被赋值

4.当全局执行环境中的代码执行前，系统会为执行环境创建一个用来管理当前执行环境中所有标识符的对象，任何在全局执行环境中定义的变量和函数都会作为global的属性存在，重复定义时，后定义的变量会覆盖前面的。

This指针：在全局执行环境中使用this时，表示global对象，在浏览器中就是window对象。

当在函数执行环境中使用this如果函数没有明显的作为非window对象的属性，而只是定义了函数，不管这个函数是否定义在另一个函数里，this都代表global使用。

**2.object对象：**

原型对象：

原型是function对象的一个属性，也就是prototype属性，每个对象都有这个内部属性，并且prototype本身也是对象。可以通过function.Prototype.属性名来给原型对象添加属性，这些属性不在构造对象里面，但他仍可以具有这个属性，这些属性就是从原型对象中继承来的，且构造函数定义的所有对象的原型只有一个。

构造函数访问属性的顺序：若构造函数有该属性，则返回构造函数的属性，若没有还属性，则会到原型中去找是否有该属性。

原型链：

\_\_proto\_\_属性指向的是构造函数的原型，在构造函数访问自身没有的属性时，就会通过\_\_proto\_\_的指向，在其指向的对象中查找。

原型链的终端：object.prototype;是所有对象的原型链的终端。

Object上的常用方法：

Object.isExtensible(obj) //obj是否可拓展（可拓展：可以在添加属性）

Obejct.preventExtensions() //使obj变成不可拓展对象并返回该对象，此方法调用之前添加的属性有效

Object.seal(obj) //封闭对象并返回该对象（对象不可拓展，属性不可配置），且封闭后不可解封

Object.isSeal(obj) //检测对象是否封闭

Object.freeze(obj) //冻结对象并返回该对象（在封闭对象的前提下，使对象所有属性变为只读，但是**setter依然可以设置属性值**）

Object.isFreeze(obj) //检测对象是否冻结

**Function对象：**

在使用function关键字定义一个函数的时候，在系统内部的实际上是创建一个对象实例，也就是说函数也是对象。

Apply方法调用函数：其参数有两个：this参数（参数传给this），以及参数数组

Call方法：其第一个参数为this参数，其他为函数参数

1.函数编译执行环境：在全局执行环境开始执行代码前会创建一个GO对象（无法访问，只会生成一个），在函数执行前，也会生成相应的AO（无法访问）对象，用于保存相应的变量及函数声明，每个函数都会生成不同的AO对象，当函数访问自身属性时，会先从AO对象中找，如果没有找到就会到相应的执行环境生成的对应的AO对象中查找，直到找到GO对象。

AO/GO对象生成分四步：

1：生成对象

2：将其中定义的变量变成属性并存储

3：将形参于实参对应

4：将定义的函数变成属性并存储

2.argument对象：函数执行前会创建一个argument对象用于管理函数的实际参数。

caller：caller是函数对象本身的属性，它显示函数的调用者，如果在全局被调用，返回null，如果在其他函数中被调用，则返回调用它的函数

callee：是argument对象属性，返回正在执行的函数，也就是函数本身。

function.bind()和function.call()两者都可以改变函数的this指向，但是两者存在不同，bind()永久改变this指向，并且传入的参数下次调用该函数是不必再传入，返回的是一个新函数，而call只是该次执行函数时this指向改变，并且会直接执行函数

3.构造函数：

New操作符来调用构造函数来创建对象时，会经过下列一系列操作

1.创建一个本地对象O

2.设置O的class属性为OBJECT或者其他本地属性：number，string，等

3.设置O的prototype属性为object.prototype或者其他对象的原型

4.调用F的call属性（调用函数constructor），将O作为this的值

5.如果call的返回值类型是对象，返回这个对象

6.返回O

实例对象的构造函数是指最原始的那个原型函数，函数对象的构造函数都是function Function() { [native code] }。函数的prototype(原型)是指最原始的原型，实例对象的prototype属性都为undefined，实例对象的\_\_proto\_\_指向构造函数的(prototype)原型

document.write(obj);调用的是obj.toString(),如果obj没有toString,则会报错

**Array对象：**数组对象

删除元素：delete arr[n]--删除第几个元素，但是数组长度并不改变

splice(m,n)--从第m位开始删除n位元素

**forEach**(function (ele, index) {}): 循环遍历数组中每个元素参数为一个函数，forEach会给函数传入两个参数，一个是数组中的元素，另一个是元素的索引

var arr = **filter**(function (ele, index) {}), 参数与forEach相同，作用是function中传回的是true则返回元素，否则不返回

Math对象：  
Math.tan()

Math.sin()

Math.cos()

Math.aqrt()----计算平方根

Math.pow(n,m)----计算n的m次方

Math.abs()----计算绝对值

**Date对象：处理日期和时间**

( New Date() ).getDate(): 返回一个月中的某一天

Date.now()返回当前到计算机元年的毫秒数（也可以用+new Date(),使用+对new Date()返回值进行转换）

.getDay(): 返回一周中的某一天

.getMonth(): 返回月份（0-11月，要获取当前月份要+1）

.getFullYear(): 返回当前的年份

.getHours(): 返回Date对象的小时（0-23）

.getMinutes(): 返回Date对象的分钟（0-59）

.getSeconds(): 返回Date对象的秒数（0-59）

.getMillseconds(): 返回Date对象的毫秒数（0-999）

.getTime(): 返回计算机元年（1970.1.1）至今的毫秒数

**ES6-Map, Set:**

Map: 字典，加强的对象

map.set(key, value) 设置属性

map.get(key) 获取属性

map.has(key) 是否含有某属性

map.delete(key) 删除

map.clear() 清空对象

map.forEach((value, key, map) => {})

map[Symbol.iterator](): 迭代器

map初始化：new Map() 参数位一个二位数组，数组每个元素位一个两个元素的数组，分别表示key value

map相对于object，map属性名不同类型之间不会相互转换，而obj[‘1’] = 2，obj[1]会替换之前值， 而map不会发生类型转换

map可以用任意原始值或者引用值做key，当引用值的地址不同被视为不同值

Set：存储任何类型的唯一值，即只有keys，访问value时相当于访问keys

set.add(key) : 添加不存在的值，如果值已存在则set不做变化

set.clear(): 清空set

set.delete(key): 删除set中某个key

set.entries, values(): 迭代器方法

set.has(key) : 是否含有某值

set[Symbol.iterator](): 迭代器

**迭代器:**

每个迭代器都有next方法，每次执行next方法，返回一个对象{value: , done: }

value每次迭代的值，done表示迭代是否结束(true or false)

每次迭代有专用指针，开始时指向数据的第一个值，每次调用next方法，返回当前值，指针下移

每个迭代器都支持for of循环，利用的时迭代器对象，迭代每个迭代器对象

所有迭代器都有的Symbol.iterator方法，必须通过中括号访问，返回默认的迭代器

原理 =》 数组中迭代器的指针，每次指向下一个索引

数组的Symbol.iterator方法调用了数组的values迭代器

Map的Symbol.iterator方法调用了Map的entries迭代器

Set由于指储存key值，所以迭代器values值仍然是keys，Set的Symbol.iterator调用了keys迭代器

所有连续储存的结构都是可以迭代的，而Object.keys(), .values(), .entries()只是从对象中获取相应的值， 并保存起来，并非是迭代器

面向对象思想 =》将所有属性及方法进行封装，并将接口暴露出去，对象属性的存储并不是连续的，所以 对象是不可迭代的

for in循环和for of 循环不同之处：for in循环按字节查找，for of循环按照存储块查找，所以for in循环的局限性就是只能迭代单字节的文本，而for fo 循环可以迭代多字节的文本

**生成器：**

定义生成器：通过function\* 和 一或多个yield语法来完成

function\*表明这是一个构造器，yield可以将yield后面的值返回，并暂停代码执行，直到调用next方法 在继续执行

注意：yield只能在构造器的直接作用域中使用，不能在其子作用域中使用

构造器中遇到return时，后面的yield也同样不会执行，并且return的值为{value:　return值， done: true}

yield可传入参数作为上一次yield的返回值

yield\* value ：可以调用value的迭代器进行迭代

**BOM-------浏览器对象模型**

**Window对象**：是BOM模型中的顶级对象，包括浏览器矿口的属性和行为

1. 控制窗口大小及位置：

moveBy(m,n)：使窗口向指定方向移动

mvoeTo(m,n)：向指定位置移动

resizeBy(m,n)：使窗口向指定方向偏移

resizeTo(m,n)：使窗口改变为指定大小

1. 用户交互的窗口：

alert(“”)：弹出窗口并显示内容，仅提示

confirm(“”)：弹出窗口并显示内容，让用户确认是否执行操作，可以取消

prompt(“提示内容”,”初始填写内容”)

1. 打开新窗口：

open(“地址”,”新窗口名”,”窗口属性”,”是否替换原窗口”)；

1. 全局函数：

setTimeout(function,time)：允许函数执行延迟

setInterval(function,time)：一段时间调用一次函数

5. <iframe></iframe>内敛框架，在页面取出一块并在里面重新打开一个页面，可设置宽高

6. 求可视区窗口（不包括控制台）：window.innerWidth(Height)(IE8及以下不兼容)

标准模式：document.documentElement.clientWidth(Height)

怪异模式：document.body.clientWidth(Height)

**Location对象：**location对象用来管理当前打开窗口的url信息

**DOM文档对象模型**

遍历**所有**节点信息：

**nodeType:** 节点类型🡪元素节点(1) 属性节点(2) 文本节点(3) 注释节点(8) document(9) DocumentFragment(11)

nodeValue: 节点值，例如标签中间的字符串

nodeName: 节点名，包括文本节点

parentNodes: 当前结点的父节点的引用

**childNodes: 当前结点的所有子节点**

firstNodes: childNodes的第一个节点

lastNodes: childNodes的最后一个

nextSibling: 下一个兄弟节点

previousSibling: 上一个兄弟节点

遍历**元素节点**树：

parentElement: 返回当前元素的父节点（IE不兼容）

**children：返回当前元素的元素节点**

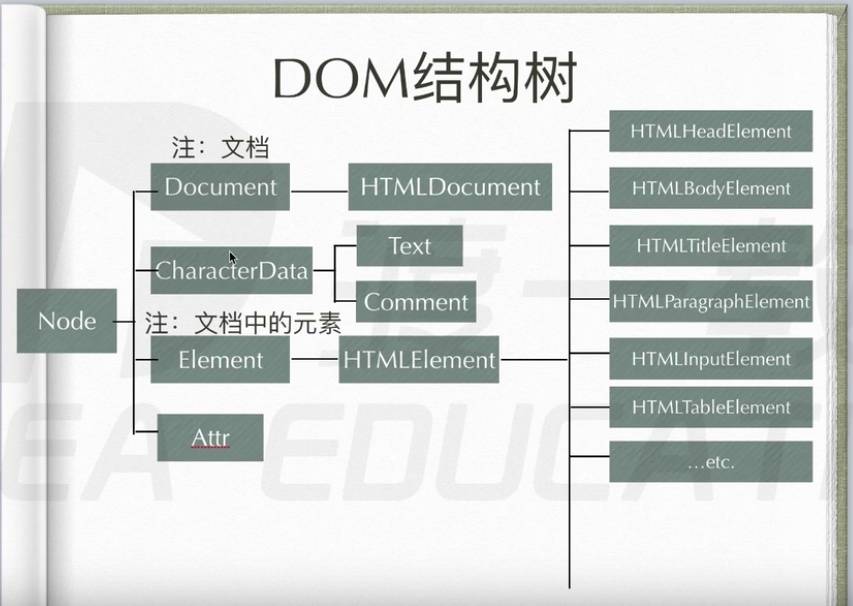
node.childrenElementCount==node.children.length: 当前元素节点的子元素

firstElementChild: 返回第一个元素的节点（IE不兼容）

lastElementChild: 返回最后一个元素的节点（IE不兼容）

属性节点：attribute()节点的属性节点的集合

**DOM结构树：**



节点的增删改查：  
增：document.createElement(‘node’) 创建元素节点

document.createTextNode(‘’) 创建文本节点

document.createComment(‘’) 创建注释节点

插入：Node.appendChild(‘’) 添加节点（可以是文本节点，也可以是注释节点）

parentNode.insertBefore(a,b); 把a添加到parentNode下的b标签前面

删：parentNode.removeChild() 将目标节点剪切出来

Node.remove() 将自身直接删除

替换：ParentNode.replaceChild(a, b) 将a和b替换

元素节点的一些功能：

Node.innerHTML=”” 替换Node里面的内容，也可以以字符串形式写入html结构

Node.innerText 替换Node里面的内容为文本(火狐不兼容)/textContent(IE不兼容)

Node.setAttribute(“属性名”,”属性值”) 设置Node节点里面的属性

Node.getAttribute() 获得节点的某个属性

Ndoe.className = “” 用来设置标签的class

Node.id = “” 用来设置标签的id

Node.getBoundingClientRect();可以获得元素的几何属性

**事件绑定**

绑定事件有三种方法：

1.obj[‘on’+ 事件] = 执行函数

2.obj.addEventListener（事件，function () {}，false）

3.IE专用：obj.attachEvent(“on” + 事件，function() {})



解除事件绑定：

1. obj[‘on’+ 事件] = null

2. obj.removeEventListener（事件，函数引用，false） 此处只能是函数引用，不能是匿名函数

3．obj.detachEvent(“on”+ type,function)

！若绑定匿名函数，则无法清除

**取消冒泡事件：obj.stopPropagation();IE9以下不支持**

**event.cancelBubble = true; IE独有**

阻止默认事件：

1.默认事件，并在结尾return false；此方法兼容性相当好，但是只有第一上列第一种绑定事件方法可用

2.event.preventDefault(); IE9以下不兼容

3.IE9以下可用event.returnValue = false;

事件

鼠标事件：

mouseover:

mousedown:

mouseup:

click:

键盘事件：键盘事件绑定在document上

! 键盘事件String.fromCharCode方法可以使unicode码转换成相应的键盘符；

事件参数e的几个属性：e.pageY:相对于文档 e.clientY: 相对于窗口 e.screenY: 相对于屏幕

文本事件：

input：当输入或删除内容时，都会触发该事件

change:当输入框聚焦和失去焦点内容不一样，在失去焦点时触发事件

focus:当获得焦点

blur:当失去焦点

**正则表达式RegExp**

创建方法：

1.var reg = /abcd/mig; //mig为正则表达式的三个功能，不分前后

2.var reg = new RegExp(“”, “”) //第一个参数是字符串，第二个参数为功能

**方法：**reg.lastIndex返回reg匹配的游标，可以手动赋值将其改变。当有修饰符g时，lastIndex才会向后移动

str.replace(a,b)用b替换a，只能替换最开始的一个，如果是str.replace(reg,str),且reg有修饰符g，则会全部替换，reg中的子表达式在b中可以用$符号引用,$1表示第一个子表达式。b还可以为function,且系统会自动传n个参数，1：正则表达式所匹配到的内容，2：第一个子表达式的反向引用……

( )\1\1\1: ( )子表达式，\1反向引用第一个子表达式（\2表示反向引用第二个子表达式），若括号内为\w，则后面会和括号内一样

修饰符：i : 不分大小写

g : 全局匹配

m : 多行匹配

**正则表达式填写规则**：

可以表示为/[][][]/每一个中括号表示为一位，里面可以放置范围，将会选中其中的任一位

abc|bcd: 表示abc或者bcd

^a在中括号外面表示行开头的a，在中括号里面表示非a

\w ：表示[0-9A-z\_] \W ：表示[^\w]

\d ：表示[0-9] \D ：[^\d]

\s ：空白字符，包含\t \n \r \v \f \S ：[^\s]

\b ：单词边界，一串字符的头前面或者位后面 \B :非单词边界

. 除换行符和其他Unicode行终止符之外的任意符号

**量词：**

n+: 1到多个n字符重复

n\*: 0到多个n字符重复

n?: 0到1个n字符

n{X}: x个n字符

n{X,Y}: X到Y个n字符

n{X,}: 最少X个n字符

n$: 在结尾的n字符

?=n: 正向预查，/a(?=b)/b参与修饰，但不参加匹配 (?!n相反)

?: ：表示忽略子表达式，参与匹配但是不显示不配结果

量词？：表示尽量少的匹配结果，即取消贪婪匹配

\n: 反向引用第n个子表达式（用小括号括起来的字符串）

正则使用注意点：

正则表达式有多个子表达式时，若用 | 连接时，只有当前面的子表达式不能匹配时，才会用其他的子表达式继续当前位置匹配

在中括号中所有的特殊字符会失效，如果要取特殊字符，例如：\*，[\*]或者\\*

/^\d+$/

reg.test(str); 将reg放到str中匹配，返回值为true或false

str.match(reg); 匹配reg并返回匹配到的结果,返回值，若是全局匹配：[第一个自表达式匹配的值，第二个自表达式匹配值，…]，若不是全局匹配：[匹配结果，匹配的位置index，执行匹配的字符串]

reg.exec(str); 返回一个数组：[匹配的字符串，第一个子表达式匹配的结果，第二个子表达式匹配的结果]。 reg.lastIndex: 返回当前匹配到的位置，与reg.test相同，需要用到reg.lastIndex,匹配后lastIndex改变为当前匹配的位置，并且每次匹配都会记录，循环后移，reg.lastIndex可以自动设置

str.search(reg); 查看字符中是否含有符合条件的字段，如果有返回位置，如果没有就返回-1

str.split(reg): 切割字符串，返回一个数组，包含切割下来的所有字符串

str.replace(reg, str1): 替换字符串

new RegExp(reg): 表示创建一个与reg相等的新对象

RegExp(reg): 返回reg的引用，即创建的对象和reg为一个对象

**HTTP模块：**

src属性：当需要加载该资源时，浏览器会向url发送GET请求

iframe: 脚本首先把发送给web服务器的信息编码到URL中，然后设置iframe的src为URL。服务器能创建一个包含响应内容的HTML文档，并把它返回给Web浏览器，并在iframe标签中显示它，脚本能够通过遍历iframe的文档对象来读取服务器的响应。这种访问受到同源策略限制

**http请求：**

**三次握手建立连接**：

1. client发起请求 --> 请i去服务是否准备就绪

2. server接受请求 --> 服务器已经准备就绪，回传消息状态给client

3. client获取到服务器就绪的消息 --> 准备就绪，client向server发送请求

**四次挥手断开连接**：

1. client发起请求 --> 请求告知服务器，准备断开连接

2. server接受请求 --> 服务器准备断开连接

3. server断开连接 -->

4. client断开连接

http链接是基于TCP协议的链接，还有一种UDP链接，两者的区别：

TCP：确认链接后才开始发送数据，不容易丢包

UDP：只要知道目标地址，不管对方是否接受链接，直接发送数据包，容易丢包，但是实时性高

**http请求由四部分组成：**

1. http请求方式或者‘动作’

2. 正在请求的URL

3. 一个可选的请求头集合， 其中可能包括身份验证消息

4. 一个可选的请求主体



**服务器返回的http响应包括3部分：**

1. 一个数字和文字组成的状态码，用来显示请求的成功和失败

2. 一个响应头集合

3. 响应主体

请求头：

**Host**：ss1.bdstatic.com //host是主机的意思，在这里表示我们请求的网址（域名地址端口号）

**User-Agent**: Mozilla/5.0(window NT 10.0; WOW64; rv:54.0) Gecko/20100101 Firefox //User-Agent是用户代理的意思，它告诉服务器客户端使用的操作系统及版本，cpu类型，浏览器及版本，主要作用是让服务器知道客户端使用的**浏览器类别**

**Accept**：text/html,application/xhtml+xml, application/xml; q=0.8 //Accept表示浏览器支持MIME（Multipurpose Internet Mail Extensions多功能Internet邮件扩展服务）类型，指定客户端能够接受的内容类型

text/html,application/xhtml+xml都是MIME类型，斜杠前面是类型，后面是子类型，类型指定大范围，子类型是类型中更为明确的类型

text是用于标准化地表示文本信息，文本信息可以是多种字符集或者多种格式

text/html表示html文档

application：用于传输应用程序数据或者二进制数据

application/xhtml+xml表示xhtml文档，application/xml表示xml文档

默认为1，若被赋值为0，则用于提醒服务器哪些是浏览器不接受的内容类型，**q是权重**

**Accept-Language**: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3

Accept-language表示浏览器支持的语言类型

zh-CN表示简体中文, zh表示中文，en-US表示美式英文，en表示英文，**q为权重**

**Accept-Encoding：**zh-CN,zh;q=0.8

浏览器能够处理的压缩编码，通常指定压缩方法，是否支持压缩，支持什么压缩方法

**Connection**：keep-alive

Connection表示客户端与服务连接类型

Keep-Alive表示持久连接

响应头：

**Allow**: 服务器支持那些请求方法

**Content-Encoding**: 文档的编码方法

**Content-length:** 内容长度

**Content-type**: 后面的文档属于什么MIME类型

**Date**: 当前GMT时间

**Expires**: 指明应该在什么时候认为文档已经过期

**Last-Modified**：文档最后改动时间

**Location**: 表示客户应当到哪里去提取文档

**Refresh**: 表示浏览器应该在什么时间之后刷新文档

cookie sessionStorage localStorage

1、cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递，而session

Storage和localStorage不会自动把数据发送给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。

 2、存储大小限制也不同，cookie数据不能超过4K，同时因为每次http请求都会携带cookie、所以cookie只适合

保存很小的数据，如会话标识。sessionStorage和localStorage虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大

 3、数据有效期不同，sessionStorage仅仅在当前浏览器窗口关闭之前有效；localStorage始终有效，窗口或者

浏览器关闭之后也一直保存，因此作用持久数据；cookie，只在设置cookie过期时间之前有效，即使窗口关闭或者浏览器关闭。

 4、作用域不同：sessionStorage在不同的浏览器窗口中不共享，即使是同一个页面，localStorage在所有的同源窗口中是共享的，cookie也是在所有同源的窗口中共享的。

 5、web Storage支持事件通知机制，可以将数据更新的通知发送给监听者。

 6、web Storage的api接口使用更方便。

**JSON:**

javascript object notation (javacsript 对象表示法)

Json是储存和交换文本信息的语法类似XML

Json是比XML更小，更快，更易解析，是轻量级的文本数据交换格式

Json.parse()可以把从后端发送到前端的信息转化为对象形式

Json.stringify()可以把前端发动到后端的数据变成字符串形式

编码相关方法：

escape：是对字符串(string)进行编码(而另外两种是对URL)，作用是让它们在所有电脑上可读。

encodeURI和encodeURIComponent都是对URL进行编码，但是encodeURIComponent编码的范围比encodeURI小，encodeURI可以将url的/一起编码，编码以后就相当于失效了，但是可以当作参数来传

**浏览器线程**

js是单线程的，但是其宿主环境(浏览器)使其可以执行异步操作；

浏览器线程：js线程，GUI线程，http网络请求线程，定时触发器线程，浏览器事件处理线程

js线程和GUI线程互斥，事件处理线程以及定时器触发线程功能：监听目标事件并将其加入js线程 中

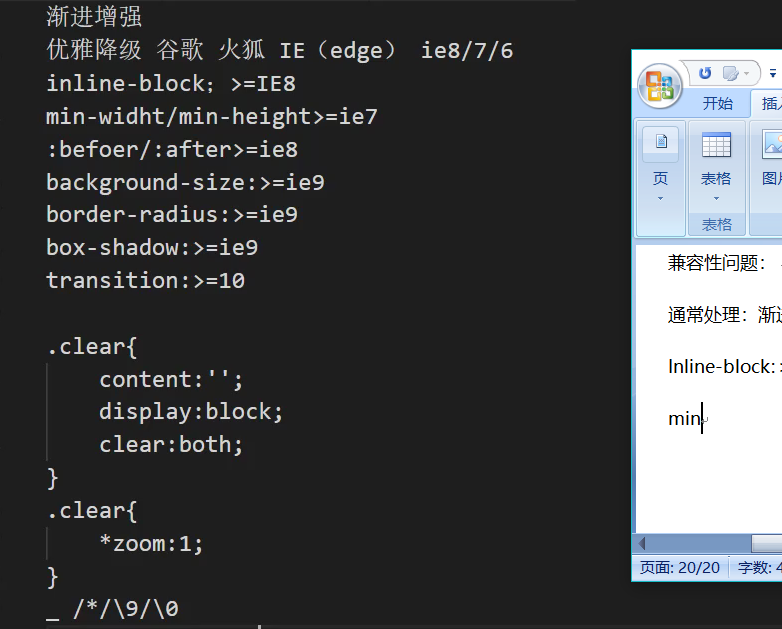
事件轮询：当任务队列中的所有任务都执行结束后，重新访问任务栈

定时器事件触发：定时器delay表示多长时间以后将事件推入事件队列，而不是直接执行（由于定时器不 需要发送请求，所以定时器事件是直接推入任务队列的）

重点：同步任务会在会加入主线程的执行栈（event stack），异步任务会进入事件表（event table），当请求结束对应的线程会将要执行的任务放进任务队列（event queue）中（定时器不同），当主线程执行栈中任务执行结束，就会访问任务队列，当任务执行结束，重新访问执行栈

兼容性问题：

通常处理：渐进增强，优雅降级



Inline-block: >=IE8

Html5shiv

Respond.js

Normalize.css

Modernizr.js

Postcss

S

caniuse.com查css兼容性